Assignment 2

Spring '24 - Anushree Kolhe

In this assignment, we would love you to explore the dataset c.elegans, a dataset of C. Elegans' neural network. It would be an excellent opportunity to get familiar with Networkx.

In [1]:
```
import networkx as nx
import pandas as pd
import os
import matplotlib.pyplot as plt
```

First, let's start reading the data. Remember that every time you read network data, make sure the type of networks you want to create. The functions provided for networks are different depending on their types. In this assignment, the one we will explore is directed weighted network, `class nx.Digraph`.

There is the canvas link (https://iu.instructure.com/files/114453534/download?download_frd=1) for downloading c.elegans. Or you can access the data through the FirstCourseNetworkScience (https://github.com/CambridgeUniversityPress/FirstCourseNetworkScience) git repo.

Let's take a look at the data first before we decide which method for reading data.

In [2]:
```
current_directory = os.getcwd()

# move back a directory
parent_dir = os.path.dirname(current_directory)
```

In [3]:
```
# make sure to repalce the path of the data when you run the code
edges = pd.read_csv(parent_dir + '/datasets/celegansneural/celegansneural.edge
```

In [4]:
```
edges.head()
```

Out[4]:

	#	Directed	weighted
0	135	1	3
1	202	1	2
2	1	2	1
3	8	2	1
4	17	2	6

In [5]:
```
len(edges)
```

Out[5]:
```
2345
```

Now we see the structure of edge data, where the first two columns refer to nodes and the third column refers to the weight of the corresponding edge. It should be noted that the first column stores the outcoming nodes, while the second column stores the incoming nodes.

In [6]:
```
G = nx.read_weighted_edgelist(parent_dir + '/datasets/celegansneural/celegansn
                              create_using = nx.DiGraph())
```

It should be noted that the type of node name, in this case, is a string instead of int.

In [7]:
```
G.has_edge(135,1)
```

Out[7]:
```
False
```

In [8]:
```
G.has_edge('135', '1')
```

Out[8]:
```
True
```

In [9]:
```
G.has_edge('1', '135')
```

Out[9]:
```
False
```

Now you have the network object, let's start exploring it a little bit. Please answer the following questions. Interpretation of the result would be appreciated. :)

Q1. How many nodes and links are in the network G?

In [10]:
```
nodes = G.number_of_nodes()
print(f"Number of nodes: {nodes}")
links = G.number_of_edges()
print(f"Number of edges/links: {links}")
```

```
Number of nodes: 297
Number of edges/links: 2345
```

Q2. What is the density of network G?

In [11]:
```
density = round(nx.density(G), 4)
print(f"Density of the network: {density}")
```

```
Density of the network: 0.0267
```

Q3. Please show the node with the largest in-degree, and the node with the largest out-degree.

In [12]:
```
def max_degree(G, degree_type) :
    if degree_type == 'in':
        degrees = G.in_degree()
    elif degree_type == 'out':
        degrees = G.out_degree()
```

```
    node, max_degree = max(degrees, key=lambda x: x[1])
    return(node, max_degree)
```

In [13]:
```
max_degree(G, 'in')
```

Out[13]:
```
('45', 134)
```

In [14]:
```
max_degree(G, 'out')
```

Out[14]:
```
('3', 39)
```

Q4. Please show the node with the largest in-strength, and the one with the largest out-strength.

In [15]:
```
def max_strength(G, degree_type) :
    if degree_type == 'in':
        strength = G.in_degree(weight='weight')
    elif degree_type == 'out':
        strength = G.out_degree(weight='weight')
    node, max_strength = max(strength, key=lambda x: x[1])
    return(node, max_strength)
```

In [16]:
```
max_strength(G, 'in')
```

Out[16]:
```
('45', 1700.0)
```

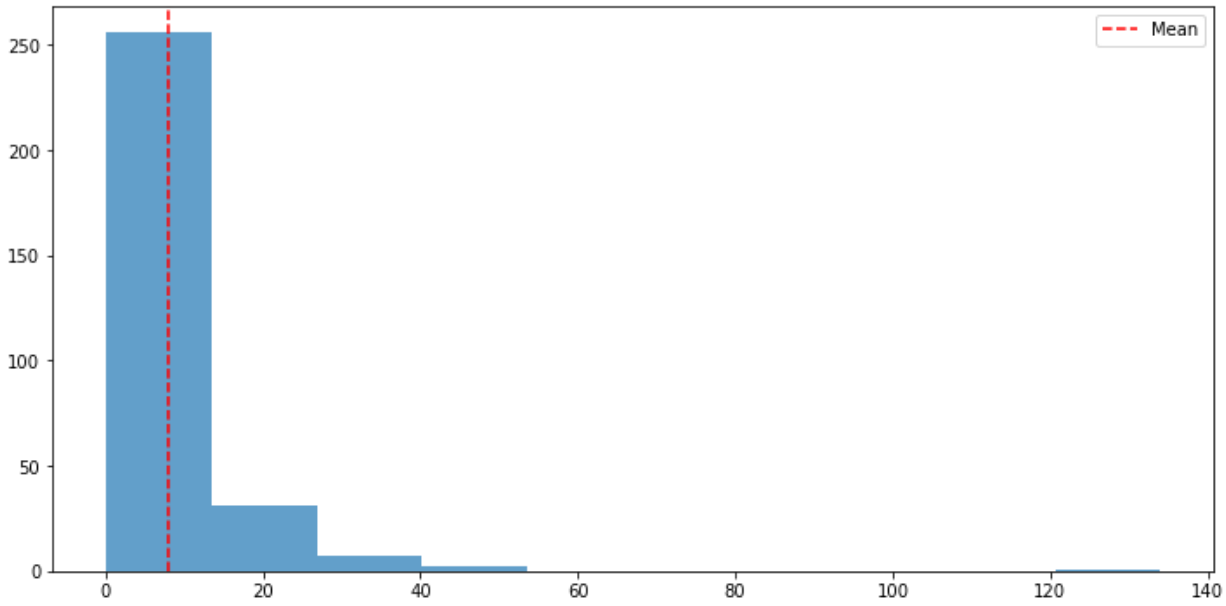
In [17]:
```
max_strength(G, 'out')
```

Out[17]:
```
('3', 197.0)
```

Q5. Please calculate the average in-degree and out-degree of the network G. Briefly interpret your finding.

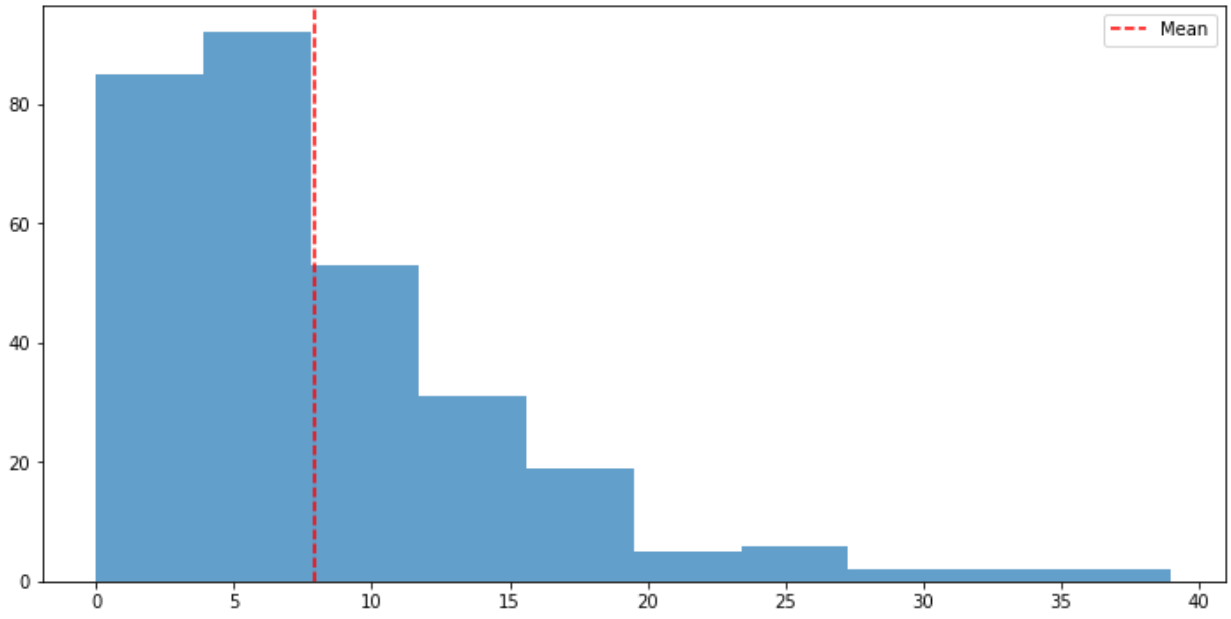
In [18]:
```
def avg_degree(G, degree_type) :
    if degree_type == 'in':
        degrees = G.in_degree()
    elif degree_type == 'out':
        degrees = G.out_degree()
    node_sum = sum(int(d) for n, d in degrees)
    avg_degree = round(node_sum/nodes, 4)

    plt.figure(figsize=(12,6))
    plt.hist(dict(degrees).values(), bins=10, alpha=0.7)
    plt.axvline(x=avg_degree, color='red', linestyle='dashed', linewidth=1.5,
    plt.legend()
    plt.show()
    return(avg_degree)
```

In [19]:
```
avg_degree(G, 'in')
```



Out[19]: 7.8956

In [20]:
```
avg_degree(G, 'out')
```



Out[20]: 7.8956

Observation :

- The average in-degree and the average out-degree of the network are the same. The mean of the degrees(in/out) for all the nodes, turns out to be the same even though the spread of the degree values is different for the in-degree and out-degree.
- Here, some nodes have high out-going links and some have high in-coming links. This might be one of the reason for observing such a pattern.

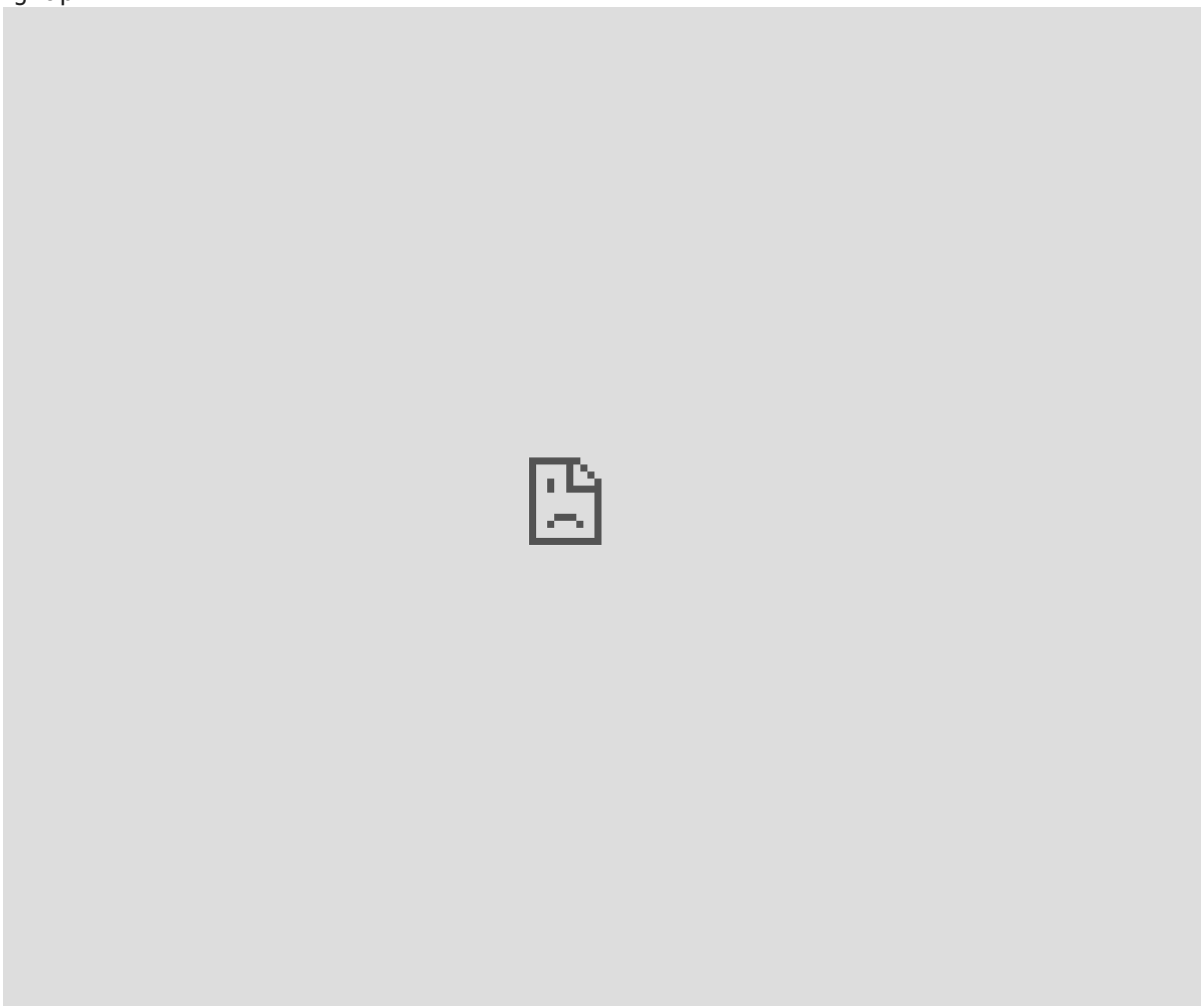
Q6. Draw the network.

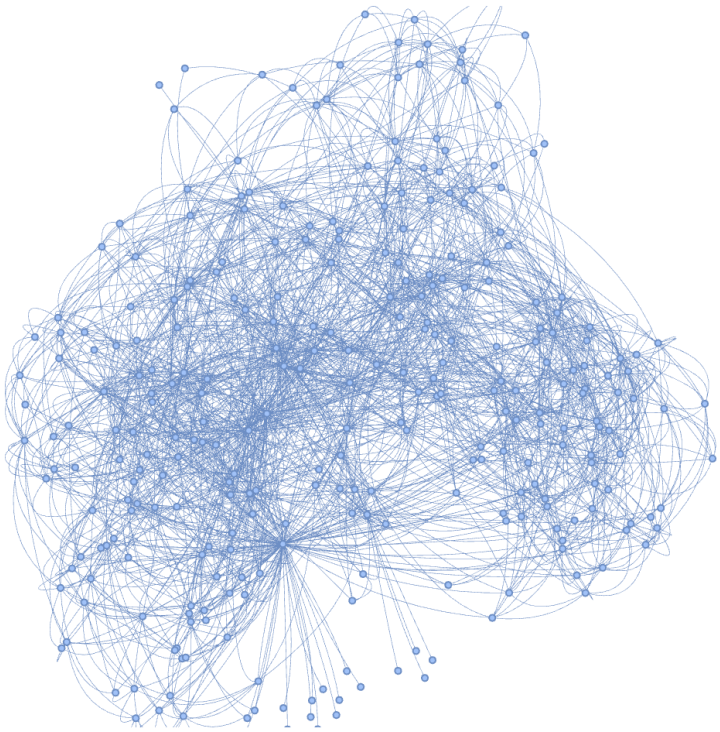
In [21]:
```
# !pip install pyvis
```

In [37]:
```
from pyvis.network import Network

# Interactive viz - Please refer html file if not visible
ntw = Network(notebook=True, width="100%", height="500px")
ntw.from_nx(G)
# ntw.toggle_physics()
ntw.show("graph.html")
```

```
Warning: When  cdn_resources is 'local' jupyter notebook has issues displaying graphics on chrome/safari. Use cdn_resources='in_line' or cdn_resources='remote' if you have issues viewing graphics in a notebook.
graph.html
```

Out[37]:





Q7. Are there any nodes with zero in-degree? How many?

In [24]:
```
def check_zero_degree(G, degree_type) :
    zero_degree = 0
    if degree_type == 'in':
        degrees = G.in_degree()
    elif degree_type == 'out':
        degrees = G.out_degree()
    zero_degree = sum(1 for n, d in degrees if d == 0)
    return zero_degree
```

In [25]:
```
check_zero_degree(G, 'in')
```

Out[25]:
```
27
```

Q8. Are there any nodes with zero out-degree? How many?

In [26]:
```
check_zero_degree(G, 'out')
```

Out[26]:
```
3
```

There comes the fun parts. Time to explore common neighbours.

Q9. Which pair of nodes have the largest number of common predecessors?

In [27]:
```
def common_neighbours(G, neighbour_type):
    max_common = 0
    nodes_max = []

    def get_neighbours(node, neighbor_type):
        if neighbor_type == 'pred':
            return set(G.predecessors(node))
        elif neighbor_type == 'succ':
            return set(G.successors(node))

    for node1 in G.nodes():
        for node2 in G.nodes():
            if node1 != node2:
                common_nodes = get_neighbours(node1, neighbour_type).intersect
                common_count = len(common_nodes)
                if common_count > max_common:
                    max_common = common_count
                    nodes_max = [node1, node2]

    return(neighbour_type, max_common, nodes_max)
```

In [28]:
```
common_neighbours(G, 'pred')
```

Out[28]:
```
('pred', 21, ['3', '119'])
```

Q10. Which pair of nodes have the largest number of common successors?

In [29]:
```
common_neighbours(G, 'succ')
```

Out[29]:
```
('succ', 29, ['3', '13'])
```

For the following questions, please read c.elegans as undirected network.

You can reread the c.elegans file, or try nx.Graph(G) to force the directed graph into an undirected graph.

In [30]:
```
G = nx.Graph(G)
```

Q11. What's the average path length?

In [31]:
```
round(nx.average_shortest_path_length(G), 4)
```

Out[31]:
```
2.4553
```

Q12. What's the diameter?

In [32]:
```
nx.diameter(G)
```

Out[32]:
```
5
```

Q13. Which node has the largest clustering coefficient?

In [33]:
```
max(nx.clustering(G,node) for node in G.nodes())
```

Out[33]:
```
1.0
```

Q14. What's the average clustering coefficient?

In [34]:
```
round(nx.average_clustering(G), 4)
```

Out[34]:
```
0.2924
```